Integrating Software-Architecture-Centric Methods into the Rational Unified Process

Rick Kazman
Philippe Kruchten (University of British Columbia)
Robert L. Nord
James E. Tomayko

July 2004

TECHNICAL REPORT
CMU/SEI-2004-TR-011
ESC-TR-2004-011

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Integrating Software-Architecture-Centric Methods into the Rational Unified Process

CMU/SEI-2004-TR-011
ESC-TR-2004-011

Rick Kazman
Philippe Kruchten (University of British Columbia)
Robert L. Nord
James E. Tomayko

July 2004

Software Architecture Technology Initiative

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Christos Scondras
Chief of Programs, XPK

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Acknowledgments

We thank Grady Booch, Bruce Macisaac, and John Smith from IBM Rational, and Felix Bachmann, Paulo Merson, and Linda Northrop from the Software Engineering Institute for reviewing an earlier draft of this report.

Abstract

The Rational Unified Process (RUP) is used broadly by software developers. This technical report fits the Carnegie Mellon® Software Engineering Institute's (SEI's) architecture-centric methods into the framework of the RUP. These methods include the Architecture Tradeoff Analysis Method®, the SEI Quality Attribute Workshop, the SEI Attribute-Driven Design method, the SEI Cost Benefit Analysis Method, and SEI Active Reviews for Intermediate Design. Since the key process milestone of the Elaboration Phase of the RUP is a completed architecture, the architecture-centric methods appear early in the process during the first two phases (i.e., Inception and Elaboration). This report presents a summary of the RUP and then examines the potential uses of the SEI's architecture-centric methods.

1 Introduction

For the past 10 years, the Software Architecture Technology Initiative[1] at the Carnegie Mellon® Software Engineering Institute (SEI) has developed and promulgated a series of architecture-centric methods, starting with the SEI Software Architecture Analysis Method (SAAM) [Kazman 96], continuing with the Architecture Tradeoff Analysis Method® (ATAM®) [Kazman 00], the SEI Quality Attribute Workshop (QAW) [Barbacci 03], the SEI Attribute-Driven Design (ADD) method [Bass 03], the SEI Cost Benefit Analysis Method (CBAM) [Kazman 02], and SEI Active Reviews for Intermediate Designs (ARID) [Clements 02a]. At the same time, the SEI has disseminated a wealth of architectural knowledge and practical expertise via its books [Bass 03, Clements 02b, Clements 02a] and papers. These efforts have now culminated to the point where the SEI is pursuing their integration by (1) combining related methods [Kazman 03, Nord 03] so they work more synergistically, and (2) fitting the architecture-centric methods into popular processes of software development. One of these processes is the Unified Process (UP).

The UP is often known as the Rational Unified Process (RUP) because Rational Software Corp. was the most successful vendor of UP. (The corporation was acquired by International Business Machines [IBM] in February 2003). To be specific, we will use the RUP 2003 version of UP as depicted by Kroll [Kroll 03], Kruchten [Kruchten 04], and the RUP software product [IBM 04]. References to the RUP in the remainder of this report refer to RUP 2003.

The SEI's architecture-centric methods have long demonstrated that they can shed considerable light on important characteristics of architectures and the quality attribute requirements that shape them. Until now, such considerations have been relegated to a separate "supplementary requirements" document in the RUP. Also, business drivers, long a key part of SEI methods, have just recently found a place in the RUP.

During the same period that the SEI's architecture-centric methods were being developed and tested, the RUP was being devised and promulgated. The RUP is an object-oriented development framework. It provides guidelines, templates, and examples for all aspects and stages of a software-intensive system's life cycle, although it treats software architecture obliquely.

---

[1] The Software Architecture Technology Initiative was formerly called the Architecture Tradeoff Analysis Initiative.

® Carnegie Mellon, ATAM, and Architecture Tradeoff Analysis Method are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

As shown in Figure 1, the RUP defines four phases—Inception, Elaboration, Construction, and Transition—each with their respective milestones. The names of the milestones were first seen in the article by Boehm [Boehm 96].

- The Inception Phase addresses the project's scope and objectives. At the end of this phase is the Life-Cycle Objective Milestone.
- The Elaboration Phase addresses major risks, builds an architecture, and evolves project plans. At the end of this phase is the Life-Cycle Architecture Milestone.
- The Construction Phase addresses detailed design, implementation, and testing. At the end of this phase is the Initial Operational Capability Milestone.
- The Transition Phase addresses fine-tuning functionality, performance, and overall quality. At the end of this phase is the Product Release Milestone.

The phases are made up of iterations of core disciplines associated with the key technical activities of software development: requirements, analysis and design, implementation, testing and assessment, deployment, and some other disciplines that are supporting in nature (configuration management, change management, project management, and the provision of an appropriate environment). We concern ourselves with the technical core disciplines in this report.

Jacobson [Jacobson 99] refers to the Unified Development Process (as he called the RUP at that time) as "iterative," "use case driven," and "architecture-centric." We briefly provide examples of each of these characteristics.

The iterative nature of the RUP is illustrated in Figure 1. The software is "grown" in a set of small iterative activities, not merely "developed" in one shot. The weight of the various disciplines (enumerated on the left side of Figure 1) is different in each phase and each iteration. This iterative process permits different things to be implemented in each pass through the disciplines. For instance, about 80 percent of the use cases are developed by the end of the Elaboration Phase after several iterations, leaving some requirements (captured eventually as added use cases) for later phases. Thus, change, especially additional requirements, is recognized as a needed and natural part of software development.

The RUP is a use-case-driven process. Use cases are used to express the functional requirements of the system in a way that is understandable to the stakeholders.

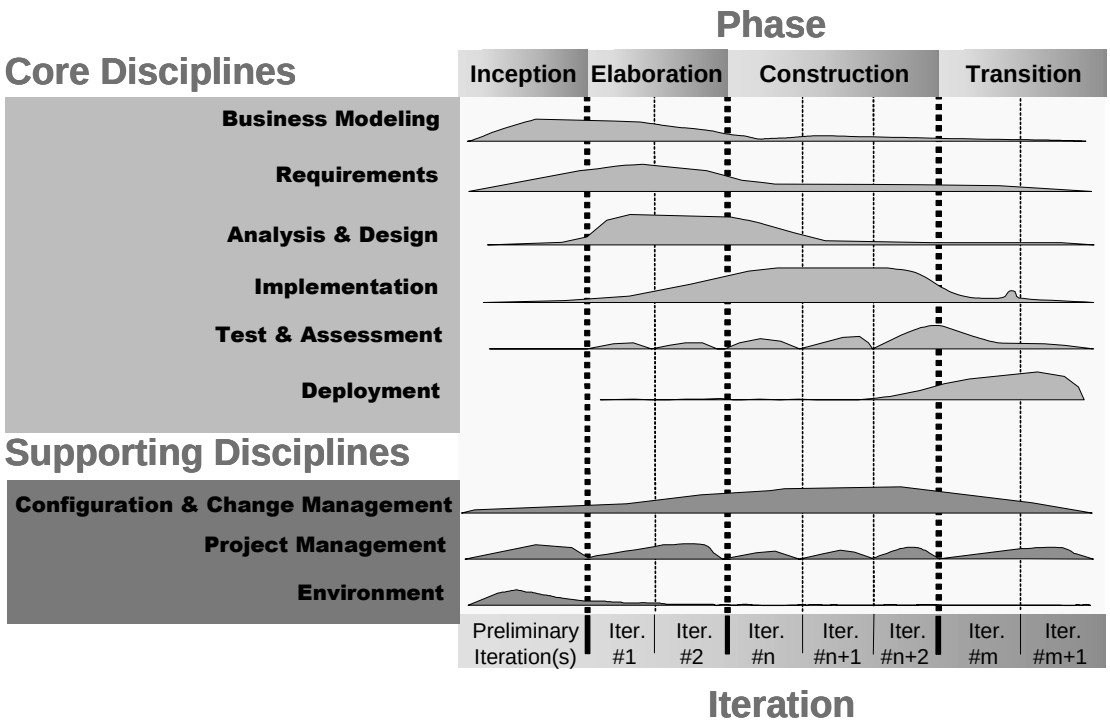

Figure 1: The Unified Process[2]

A final important point to be made about the RUP is that it is architecture-centric. There is no definition of this fact in the RUP literature (except for the architecture development chart in the book by Kruchten [Kruchten 04]), so we use the list of characteristics of architectural centricity developed by Bass, Clements, and Kazman [Bass 03] and reproduced in Figure 2. In the RUP, Jacobson and Kruchten define an architecture definition milestone by the end of the Elaboration Phase [Jacobson 99, Kruchten 04]. This means that much of the architecture is designed and analyzed in the Elaboration Phase, so as to be ready for the Construction Phase. For this reason, all SEI architecture-centric methods belong in the first two phases of the RUP: Inception and Elaboration.

[2] Reproduced by permission based on material from Rational Unified Process, Version 2003.06.01.04, © Copyright 1987-2003 by International Business Machines Corporation. All rights reserved.

Architecture-centric development involves iteratively

- creating the business case for the system
- understanding the requirements
- creating or selecting the software architecture
- documenting and communicating the software architecture
- analyzing or evaluating the software architecture
- implementing the system based on the software architecture
- ensuring that the implementation conforms to the software architecture

Figure 2: Life-Cycle Activities of Architecture-Centric Development

In Section 2 of this report, we look at how the RUP describes software architecture and some shortcomings related to software architecture. In this section, we also introduce a case study for an automated teller machine as a means of exemplifying the RUP in use. Section 3 looks at the core discipline of requirements identification and suggests how the Quality Attribute Workshop might play a role in the RUP. In Section 4, we look at the core discipline of analysis and design, and we examine the potential place of the ADD method in that discipline. Section 5 describes the potential role of the ATAM and CBAM at various places in the Elaboration Phase, and Section 6 describes the use of ARID within this phase. In each case, we attempt to describe all the SEI architecture-centric methods as RUP activities. Section 7 concludes this report with some reflections on the usefulness of augmenting the RUP in this fashion.

2 Software Architecture and the RUP

In this report, we use an example of an automated teller machine (ATM) to illustrate the differences between the Rational Unified Process and the SEI's architecture-centric methods and to illustrate how they might complement each other.

2.1 Example: An ATM System

Let us first examine this example purely from the perspective of the RUP. In the RUP, a systems analyst would elicit and record the required system functionality in the Inception Phase and early in the Elaboration Phase in the requirements workflow. The requirements are embodied in the use cases, the special requirements property of use cases, and the supplementary specifications.

2.1.1 Eliciting Requirements

A *use case* is a sequence of actions a system performs that yields an observable result of value to a particular actor. An actor is someone or something outside the system (typically a human, although not necessarily) that interacts with the system [Kruchten 04]. For example, use cases from the perspective of the "ATM customer" actor would include activities such as using the ATM to withdraw money, deposit money, transfer money between accounts, or check the balance of a bank account. These capabilities can be represented as use cases, as shown in Figure 3.

Of course, there will be other actors as well (such as ATM service personnel, bank branch employees, and auditors), and they will also have use cases represented in the diagram. The collection of use cases constitutes the complete functionality of the system.

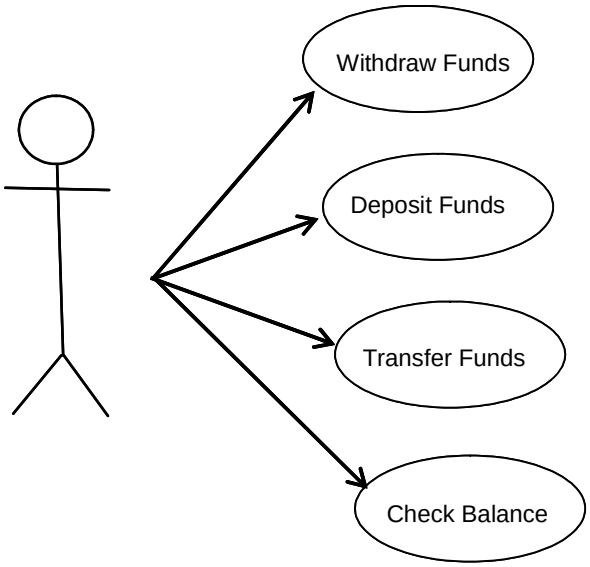

Figure 3: ATM Use Case Diagram (Notation: UML)

However, there are other important qualities of the system in addition to its functionality (for example, performance requirements specifying that the customer must get a response from the system in less than 10 seconds). There are also availability requirements stating that the system must be available 24 hours a day and 7 days a week, with the exception of a 15-minute "service time" window each day, and that the system must be able to recognize and report faults within 30 seconds of their occurrence. There are security properties dictating that the transaction must be properly authorized and communicated securely to the bank's database. In addition, there are modifiability properties dictating that the system must be easily modified to take advantage of new platform capabilities (for example, it must not be tied to a single database or to a single kind of client hardware or software) and that it must be extensible to allow the addition of new functions and new business rules.

From the collected use cases, an analysis process is entered that results in a class diagram for the system and other design models (such as sequence diagrams, activity diagrams, and state charts). But these artifacts specify primarily the system's functionality and are insufficient to specify an architecture for the ATM system that will address the system's quality attributes. These attributes, or nonfunctional requirements as the RUP refers to them, are captured in properties of the use cases as well as in the supplementary specifications.

The *special requirements property* of a use case is a textual description that collects all the requirements (such as quality requirements) on the use case that are not considered in the use case model, but that need to be taken care of during design or implementation. Performance properties such as the latency of an operation (such as an ATM withdrawal) can be captured as a timing property that annotates the use case describing the functionality of the withdrawal.

The *RUP Supplementary Specification* artifact captures system requirements that are not readily noted in behavioral requirements artifacts such as use case specifications. The supplementary specifications are an important complement to the use case model, because together they capture all the software requirements (functional and nonfunctional) that need to be described to serve as a complete software requirements specification [Kruchten 04].

2.1.2 Creating and Evaluating a Design

In the RUP, an architect or architecture team designs the architecture in the Inception and Elaboration Phases following the "Perform Architectural Synthesis," "Define a Candidate Architecture," and "Refine the Architecture" workflow details of the analysis and design discipline. An architectural proof-of-concept may be defined in the Inception Phase.

A candidate architecture (such as that exemplified in Figure 4) is produced early in the Elaboration Phase. The *candidate architecture* is an initial sketch of the system architecture that defines an initial set of architecturally significant elements to be used as a basis for analysis, an initial set of analysis mechanisms, and the initial layering and organization of the system [Kruchten 04].

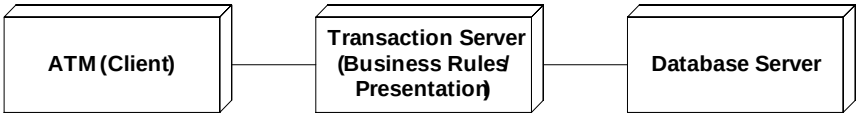

Figure 4: A Candidate Architecture – Deployment View (Notation: UML)

A candidate architecture might be posited by the architect based on his or her experience and structured using the repository and three-tiered client-server styles. The ATM is, thus, a client of a transaction-processing system.

An executable architecture is produced during the Elaboration Phase as the architecture is refined. An *executable architecture* is a partial implementation of the system, built to demonstrate that the architectural design will be able to support the key functionality and, more importantly, to exhibit the right properties in terms of performance, throughput, capacity, reliability, scalability, and other "-ilities" [Kruchten 04].

The 4+1 approach provides some guidance in terms of the artifacts that need to be produced [Kruchten 95]. Functionality is a primary driver of the architecture's structure.

The architecture is reviewed at the end of the "Refine the Architecture" workflow detail.

2.2 The ATM Example Revisited

Let us now revisit our example with the aim of showing how the SEI's architecture-centric methods can enhance the RUP. A candidate architecture, such as the (primitive) one given in Figure 4, is a product of the architecturally relevant use cases that have been identified. But this architecture is dependent, for its shape and its quality, on the experience of the architecture team. The SEI architecture-centric methods can inform and regularize this process. The QAW can help elicit quality attribute requirements in the form of quality attribute scenarios. The ADD method defines a software architecture by basing the design process on the quality attributes that the software must fulfill. The ADD method documents a software architecture in a number of views: most commonly, a module decomposition view, a concurrency view, and a deployment view [Clements 02a].[3] The ADD method depends on an understanding of the system's constraints, as well as its functional and quality requirements, which are represented as six-part scenarios. The ATAM, CBAM, and ARID provide detailed guidance on analyzing the resulting design.

2.2.1 Eliciting Scenarios

The QAW can help elicit quality attribute requirements in the form of quality attribute scenarios.[4] A *quality attribute scenario* is a quality-attribute-specific requirement [Bass 03] that consists of six parts:

1. *stimulus*: This is a condition that needs to be considered when it arrives at a system.
2. *source of the stimulus*: This is the entity (an actor) that generated the stimulus.
3. *artifact stimulated*: Some system artifact is stimulated by the stimulus. This artifact may be the entire system or some portion of it.
4. *environment*: The stimulus occurs within a specified context. For example, the system may be in a normal state, a degraded mode, or in an overload condition when the stimulus occurs.
5. *respons*e: The response is the activity undertaken upon the arrival of the stimulus.
6. *response measure:* When the response occurs, it should be measurable in some fashion so the quality attribute requirement can be tested.

Scenarios, as elicited and elaborated in the architecture-centric methods, are very similar to use cases: they indicate what must be present, what is done, and what the outcome will be. Therefore, use cases and scenarios can and should be developed simultaneously. Essentially,

---

[3] Clements and colleagues [Clements 02a] illustrate how these views relate to the 4+1 views [Kruchten 95] often used as an example within the RUP.

[4] Note that UML has its own notion of a scenario (an instance of a use case) that is used in the RUP; for the purposes of this report, we use the term *scenario* as shorthand for the quality attribute scenario.

the scenarios inspire and are inspired by use cases. The difference is that scenarios always include the six elements above and, hence, are always focused on the elicitation and documentation of quality-attribute-specific information. Scenarios may mention functionality, but that is not their point.

The point of the scenarios is that the set of elements is an embodiment of the quality attribute requirements, and it is these requirements that inspire and shape an architecture. Simply put, the architecture is determined by the quality attribute requirements, not by the functionality. Returning to our example, a quality attribute (performance) scenario that corresponds to the use case for an automated teller machine is as follows: "The user can withdraw a limit of $300 from an account that has sufficient funds in less than 10 seconds." There are two functional requirements and one performance requirement in this scenario. One function is a withdrawal, and one is a limit (a constraint of $300 if it is in the account). There's also a performance constraint of "less than 10 seconds," which is a quality attribute. Typically, a use case would not include such a performance constraint.

The SEI's architecture-centric methods provide several techniques for eliciting scenarios. The QAW, for example, is a facilitated method that engages system stakeholders early in the life cycle to discover the driving quality attributes of a software-intensive system and for recording these quality attributes in the form of scenarios in the six-part documentation structure outlined above. In addition to the brainstorming activity in the QAW, the architecture-centric methods elicit and capture quality attribute scenarios in two other ways—by using general-scenario-generation tables and utility trees. These are described by Bass and colleagues [Bass 03].

2.2.2 Creating and Evaluating a Design

One quality attribute requirement mentioned earlier is that the system must be easily modified to take advantage of new platform capabilities (such as a new database or client), and it must be extensible to allow new functions and new business rules to be added. Through the process of the QAW, this vague requirement would be refined into several six-part scenarios. For example, the following modifiability scenarios would be typical of an ATM:

- A developer wishes to add a new auditing business rule at design time and makes the modification, without affecting other functionality, in 10 person-days.
- A developer wishes to change the relational schema to add a new view to the database, without affecting other functionality, in 30 person-days.
- A system administrator wishes to employ a new database and makes the modification, without affecting other functionality, in 18 person-months.
- A developer wishes to add a new function to a client menu, without side effects, in 15 person-days.

- A developer needs to add a Web-based client to the system, without affecting the functionality of the existing ATM client, in 90 person-days.

To achieve these modifiability requirements, one or more architectural tactics will need to be employed. An architectural tactic is a means of satisfying a quality-attribute-response measure (such as average latency or mean time to failure) by manipulating some aspect of a quality attribute model (such as performance queuing models or reliability Markov models) through architectural design decisions [Bachmann 02]. In this way, tactics provide a generate-and-test model of architectural design. The ADD method defines a software architecture by basing the design process on the quality attribute requirements of the system. The ADD approach follows a recursive decomposition process where, at each stage in the decomposition, architectural tactics and patterns are selected to satisfy a chosen set of high-priority quality scenarios.

In the case of modifiability, relevant architectural tactics include *Localize Changes* and *Use an Intermediary.* The *Localize Changes* tactic suggests that the business rules, database, and client should be localized into components, and the *Use an Intermediary* tactic suggests that each of these components should be separated to insulate them from potential changes in each other. The three-tier client-server model, shown in Figure 4, would emerge from the application of the *Localize Changes* tactic, since this architecture allocates the client, database, and business rules to their own tiers and, hence, localizes the effects of any changes to a single tier. The *Use an Intermediary* tactic suggests that the communication between the tiers be mediated by some abstract interface (such as a data access layer that uses Open DataBase Connectivity [ODBC] between the business rules and the database) and a translation layer between the business rules and the client that understands the Extensible Markup Language (XML). The existence of such intermediaries makes it simple to add new databases or clients. For example, a developer can now add a Web-based client and server as a simple addition to the architecture, without affecting the ATM client.

To achieve the quality attribute requirement of a “10 second latency on a withdrawal” in the ADD method, a different set of architectural tactics are employed. Performance tactics are divided into three categories: resource demand, resource management, and resource arbitration. Since we cannot control resource demand with an ATM (or, more precisely, because doing so would be bad for business), we must look towards managing and/or arbitrating the use of resources to meet performance goals. Some resource management tactics that are potentially applicable here are *Introducing Concurrency*, *Maintaining Multiple Copies of Either Data or Computations,* and *Increasing Available Resources.* By employing the *Introducing Concurrency* and *Increasing Available Resources* tactics, we may choose to deploy additional database servers and business rule servers or to make any of these servers multithreaded so they may execute multiple requests in parallel. Once we have multiple resources, we will need some way of arbitrating among them. Thus, we choose to introduce a new component–a load balancer–that will employ one of the resource arbitration tactics, such as *Fixed-Priority*

Scheduling or *First-In First-Out Scheduling*. This component will ensure that the processing load is distributed among the system's resources according to a chosen scheduling policy.

This leads us to the design shown in Figure 5: a slightly revised and elaborated version of the architecture initially presented in Figure 4. Obviously, both of these architectures are still simple, and much more work needs to be done to turn them into complete design specifications for development. The purpose of this example is not to show the entirety of a sophisticated architecture being developed, but rather to emphasize the difference in how we arrived at the architectures of Figure 4 and Figure 5. In Figure 4, the architecture was created out of the architect's experience and knowledge. When using ADD, on the other hand, tactics and a structured set of steps provided design guidance for the creation and nature of each tier. In this way, each architectural structure is created via an engineering process, rather than simply being adopted out of habit, intuition, or experience.

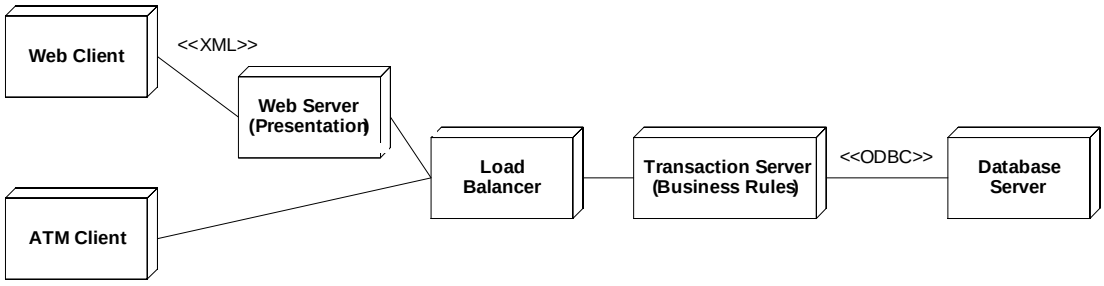

Figure 5: A Revised Candidate Architecture – Deployment View (Notation: UML)

In this view, we have not yet specified the precise degree of replication of any of the deployed clients or servers, or the size of the thread pool in each of them. This more detailed specification is the next step in the design process. Once these characteristics have been specified, the latency characteristics of the architecture can be evaluated via a performance queuing model. However, architectural decisions are complex and they interact. For example, the degree to which changes in the database schema will affect the business rules, Web server, or client software also needs to be analyzed. Each of the abstraction layers (XML and ODBC) will mask some class of changes and expose others. And each layer will impose a performance cost. Similarly, the addition of a load-balancing component will create additional computation and communication overhead, but will provide the ability to distribute the load among a larger resource pool.

It is clear that design decisions interact. For this reason, we need an organized method for understanding the interaction of the many decisions that are made in creating a complex system architecture. The ATAM provides software architects with a framework for understanding the technical tradeoffs and risks they face as they make architectural design decisions. In addition, the CBAM helps software architects consider the return on investment (ROI) of any architectural decision and provides guidance on the economic tradeoffs involved. Finally,

ARID evaluates whether the design can be used by the software engineers who must work with it.

2.3 Summary

The SEI architecture-centric methods can provide explicit and detailed guidance on eliciting the architectural requirements, designing the architecture, and analyzing the resulting design:

- The architecture-centric methods place an emphasis on quality attributes rather than functionality.
- The architecture-centric methods help fill gaps in the RUP design process by providing specific advice on
  - the elicitation and documentation of quality attribute requirements
  - which design operation will achieve a desired quality attribute response
  - how to analyze the result to understand and predict the consequences of the design decisions in terms of risks, tradeoffs, and ultimately the ROI
- The architecture-centric methods all use common concepts: quality attributes, architectural tactics, and a "views and beyond" approach to documentation that leads to more efficient and synergistic use [Clements 02b]. Note that the topic of incorporating the "views and beyond" approach to documentation as activities and artifacts into the RUP is beyond the scope of this technical report.

Next, we look at each method in more detail.

3 Requirements Identification

A Quality Attribute Workshop (QAW) can be held early in the Inception Phase to elicit and analyze the required quality attributes of any candidate architecture. To be successful, this workshop needs to gather a wide group of stakeholders from the business organization. Scenarios developed during the QAW can be further refined into use cases, and they can help to develop the test plan. The QAW contains activities found in the requirements discipline.

3.1 The QAW

The QAW is a facilitated method that engages system stakeholders early in the life cycle to discover the driving quality attributes of a software-intensive system. The QAW has its roots in, and was developed to complement, the ATAM. The QAW provides a way to identify important quality attributes and clarify system requirements before the software architecture has been created. The QAW elicits, collects, and organizes software quality attribute requirements in the form of scenarios.

Figure 6 provides a summary of the inputs, outputs, and participants of the QAW. This figure is based on a functional modeling notation [IEEE 98] where inputs flow in from the left, outputs flow out to the right, and the participants of the method are noted below. More details about the QAW are available in the report by Barbacci and colleagues [Barbacci 03].

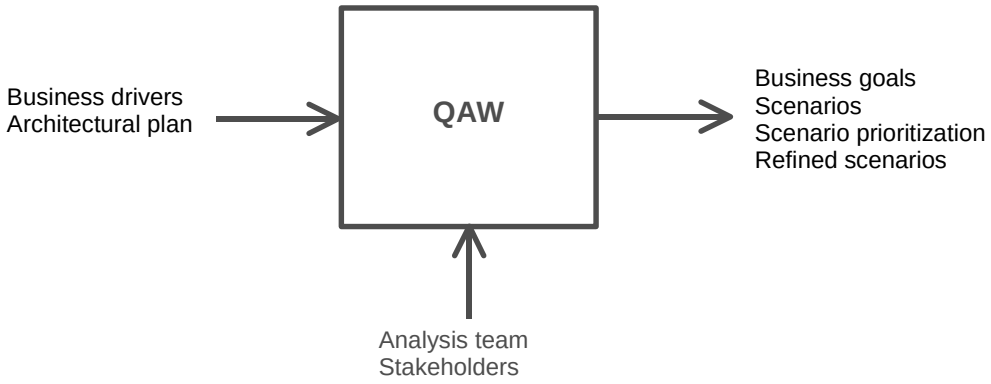

Figure 6: QAW Inputs, Outputs, and Participants

3.2 The QAW and the RUP Life Cycle

The RUP has a workflow defined for requirements that ultimately leads to the definition of the system requirements of the system gathered in the software requirements specification (SRS) artifact. The requirements task is one of the important disciplines of the RUP's Inception and Elaboration Phases. A substantial amount of effort in the Inception Phase is given to the discipline of requirements elicitation, capture, documentation, and analysis. Other important activities in the Inception Phase include creating the business case and finding a candidate architecture.

The systems analyst is responsible for the use case model and leads and coordinates requirements elicitation and use-case modeling by outlining the system's functionality and delimiting the system. The requirements specifier is responsible for the SRS and specifies the details of one or more parts of the system's functionality.

The RUP provides well-defined activities and guidelines for producing the use case model, such as those for holding a requirements workshop as a means of eliciting stakeholder requests. Later, these requests can be refined into use cases during a use case workshop. The emphasis is on functionality, but any requirements that cannot be captured as a use case are recorded and to later form the basis for the supplementary specifications.

Nonfunctional requirements are captured during the Inception Phase and early in the Elaboration Phase. There are, within the RUP, some guidelines and placeholders for the inclusion of nonfunctional requirements. Such placeholders include properties of use cases and the supplementary specifications.

- Use cases have a property called "special requirements" that is defined as "a textual description that collects all requirements, such as nonfunctional requirements, on the use case that are not considered in the use case model but that need to be taken care of during design or implementation."
- Supplementary specifications include nonfunctional requirements. One example that the RUP provides for categorizing them is with the FURPS+ model [Grady 92]. That model describes the major categories as functionality, usability, reliability, performance, and supportability. The "+" is for additional requirements such as design constraints and implementation, interface, and physical requirements.

A QAW can clearly enhance this process. A QAW would be appropriate for the Inception Phase and aid in meeting objective 2 of the Inception Phase: "Identify the key system functionality" [Kroll 03]. The QAW would thus provide input to the activities for meeting the next objective of the Inception phase: "Determine at least one possible solution."

A QAW could be held again in the Elaboration Phase as a follow-on activity to a QAW that was held in the Inception Phase. For example, a guideline could be written to hold a QAW after the requirements workshop. Such a workshop could be held in parallel with the use case workshop to provide a more focused understanding of the requirements. Or, workshops could be held for one or more subsystems in a complex system of systems. Instead of a workshop format, the activities of scenario elicitation and refinement could be performed iteratively by the requirements specifier, as the SRS is refined.

3.3 The QAW as an Activity in the RUP Requirements Discipline

The QAW could be modeled as an activity in the requirements discipline (see Figure 7). The participants, inputs, outputs, and function from Figure 6 correspond to the role, input artifacts, resulting artifacts, and steps in Figure 7. The QAW analysis team members play the role of systems analyst in the RUP.

The inputs to the QAW can come from other RUP artifacts. Business drivers come from the business vision artifact within the business-modeling discipline. The architectural plan contains information about architecture development plans, including known technical constraints such as an operating system (OS), hardware, or middleware prescribed for use; other systems with which the system must interact; key technical requirements that will drive architectural decisions; and existing context diagrams, high-level system diagrams, and descriptions (e.g., Command, Control, Communications, Computer, Intelligence, Surveillance, and Reconnaissance [C4ISR]). Some of this information comes from the RUP vision document artifact [Leffingwell 00]. Other sources of this information may include, for example, existing systems, legacy systems, and documentation of commercial off-the-shelf (COTS) products, if they are applicable. These other sources are not produced by the RUP, but rather are external inputs to the RUP process.

The outputs from the QAW feed into other RUP activities and/or refine other artifacts. For example, business goals are refined during the QAW and could be fed back to the RUP business vision. The scenarios can help determine what is in/out of the system's scope and can lead to the creation or refinement of the system context diagram or its equivalent. Scenario generation can also lead to the creation of use cases. There is no explicit RUP artifact that corresponds to QAW scenarios, but the information they contain about quality attributes can be captured as properties of use cases or in the supplementary specification.

<table>
<tr><td colspan="2">Elicit Quality Attribute Scenarios Using the Quality Attribute Workshop (QAW)</td></tr>
<tr><td colspan="2">Purpose: The Quality Attribute Workshop (QAW) is a facilitated method that engages system stakeholders early in the life cycle to discover the driving quality attribute requirements of a software-intensive system. The key points about the QAW are that it is system-centric, stakeholder focused, and used before the software architecture has been created.</td></tr>
<tr><td colspan="2">Role: Systems analyst [Analysis team]</td></tr>
<tr><td colspan="2">Frequency: This activity occurs as required, typically once per iteration in the Inception Phase and once in the Elaboration Phase.</td></tr>
<tr><td colspan="2">Steps in the QAW:
1. QAW Presentation and Introductions
2. Business/Programmatic Presentation
3. Architectural Plan Presentation
4. Identification of Architectural Drivers
5. Scenario Brainstorming
6. Scenario Consolidation
7. Scenario Prioritization
8. Scenario Refinement</td></tr>
<tr><td>Input Artifacts:
• business case [business drivers]
• vision document [architectural plan]</td><td>Resulting Artifacts:
• business case [QAW refines the business goals, which provide feedback to the business case]
• supplementary specifications [as repository of scenarios]</td></tr>
<tr><td colspan="2">Tool Mentors: None</td></tr>
<tr><td colspan="2">More Information: [Barbacci 03]</td></tr>
<tr><td colspan="2">Workflow Details:
• Requirements
  • Understand Stakeholder Needs</td></tr>
</table>

Figure 7: The QAW as a RUP Activity[5]

[5] Correspondence to SEI terms is included in square brackets.

3.4 Reflections

The RUP fills a need in the SEI's architecture-centric methods by placing the QAW in a life-cycle context. One issue that needs to be addressed is how scenarios produced in a QAW can be used by a software architecture design method such as the ADD method or used by a subsequent evaluation method such as the ATAM. The QAW scenarios tend to be at the system level, whereas the ADD method scenarios are at the software level. The RUP for System Engineering (RUP SE), which is a variant of RUP, has a notion of use case flow down [Cantor 03]. This concept might apply to scenarios, where the initial scenarios generated in the QAW are partitioned among the software, hardware, people, and data that make up the system, and those allocated to software flow down to the ADD method or ATAM.

The QAW fills a need in the RUP by providing an explicit method for gathering quality attribute scenarios from stakeholders and a six-part template for representing scenarios. The RUP has placeholders for this kind of information in the properties of use cases and the supplementary specifications. The QAW approach would favor augmenting use cases with quality attribute information in the six-part scenario format. This format could be captured as a RUP guideline for structuring what otherwise would be unstructured text descriptions of non-functional requirements. These scenarios would also be used as architecturally significant requirements when defining the candidate architecture.

Not only does this approach give the systems analyst guidelines for being more precise about the meaning of quality attributes, but it also brings more prominence to the quality attributes and their role in shaping the architecture.

In addition to the more immediate benefits cited above, the scenarios continue to provide benefits during later phases of development. Refined scenarios can be documented as sequence diagrams or collaboration diagrams to capture the behavior of elements in the architecture. Stakeholders' concerns and any other rationale information that is captured should be recorded individually in a form that can be included in the appropriate architecture documentation. Scenarios provide input for analysis throughout the life of the system and can be used to drive test-case development during implementation testing.

4 Architecture Design

The Attribute-Driven Design (ADD) method defines a software architecture by basing the design process on the quality attributes that the software must fulfill; thus it can create a "candidate architecture" as defined in the RUP. The ADD method can and should be held early in the Elaboration Phase to set the stage for a more complete architecture design. Among the exit criteria for the Elaboration Phase is a software architecture.

4.1 The ADD Method

The ADD method creates and documents a software architecture in a number of views: most commonly, a module decomposition view, a concurrency view, and a deployment view. The ADD method depends on an understanding of the system's constraints and its functional and quality requirements. Figure 8 provides a summary of the method's inputs, outputs, and participants. More details about the ADD method are available in the book by Bass and colleagues [Bass 03].

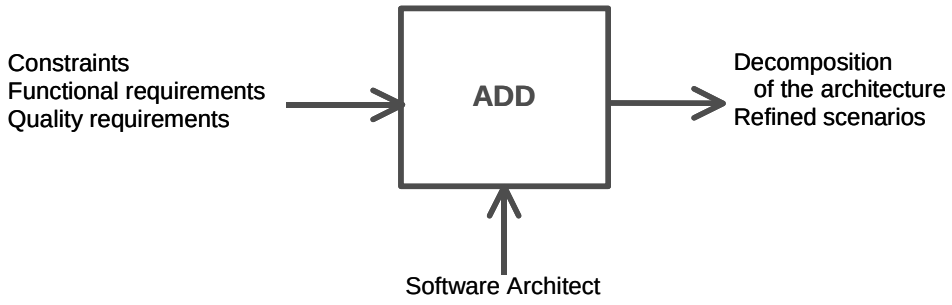

Figure 8: The ADD Method's Inputs, Outputs, and Participants

4.2 ADD and the RUP Life Cycle

The RUP has a workflow defined for analysis and design that ultimately leads to the software architecture. Architecture design is addressed primarily in the Inception and Elaboration Phases of the RUP. The RUP software architect is responsible for the software architecture, which includes the key technical decisions that constrain the design and implementation of the project. The software architect might also construct an architectural proof-of-concept during the RUP's Inception Phase. A candidate architecture is defined early in the Elaboration Phase, and later refined.

The milestone that concludes the Elaboration Phase is the Life-Cycle Architecture (LCA) Milestone. The artifacts associated with this milestone include the software architecture document organized along a set of views, chosen from the "4+1" set (logical, implementation, process, deployment, and use case) or any other suitable set of views for the project, an executable architectural prototype, and the architectural mechanisms. The architecture is used by the stakeholders and could be modified in subsequent phases. Even though the architecture is produced as part of this phase, it may be revisited in later phases as more use cases, and thus requirements, are known.

Architecture is first addressed in the RUP Inception Phase during the workflow detail "Perform Architectural Synthesis." One of the outcomes is the production of an architectural proof-of-concept, the purposes of which are to get feedback from the customer, so the team can better document what the customer wants, and to get a better understanding of the technology and associated risks. The proof-of-concept may take the form of a list of known technologies (e.g., frameworks, patterns) that seem appropriate to the solution; a sketch of a conceptual model of a solution using a notation such as the Unified Modeling Language (UML); a simulation of a solution; or an executable prototype. The team might implement bits of the technology or begin to make decisions about what to purchase.

An initial sketch of the architecture is defined early in the Elaboration Phase. This initial sketch is known as the candidate architecture. The architecture is next addressed in the RUP Elaboration Phase during the workflow detail "Refine Architecture." It is documented in the software architecture document (SAD) and implemented as an evolving executable architecture.

ADD can enhance this process; it is an activity performed by the software architect that contributes to the initial software architecture design known as the candidate architecture. It is a specific design method with a detailed set of steps aimed at producing an architecture that both satisfies the desired quality and business goals and provides the framework for realizing the desired functionality.

Activities of the ADD method can enrich some of the workflow details in the RUP analysis and design discipline, specifically, "Define a Candidate Architecture" and "Analyze Behavior." In the "Define a Candidate Architecture" detail, the focus is on identifying the architectural drivers and producing an initial structure of the architecture that satisfies these qualities. The ADD Method uses architectural tactics associated with quality attribute scenarios to help guide this activity.

In the "Analyze Behavior" detail, behavioral descriptions provided by the requirements are transformed into a set of elements. The ADD method provides steps for allocating functionality (from the other requirements) to the structure identified in the candidate architecture. The other workflow details ("Refine the Architecture," "Design Components," and "Design the Database") begin where ADD ends.

The architecture created as an output of ADD is a representation of the most important design choices. It describes a system as containers for functionality and interactions among them. Because it is the first articulation of the architecture during the design process, it is necessarily coarse grained.

The outputs of the ADD method are the first several levels of a module decomposition view of an architecture and other views as appropriate. The method typically uses the following views to document the software architecture: a *module view* that contains responsibilities of design elements and the data interactions among those elements; a *concurrency view* that contains threads of control and the synchronization relationships among design elements; and a *deployment view* that shows the allocation of design elements to processors. These views are described in the "views and beyond" approach to documenting software architectures [Clements 02a], which also describes a process for choosing appropriate views based on the stakeholders and their documentation needs.

During the subsequent design steps (which are not part of the ADD method), the containers for functionality are specified in more detail. These steps involve producing a more formal specification of the behavior of containers (modeled by state charts and interaction diagrams) and a more formal specification of interactions (signatures and protocols).

4.3 ADD as an Activity in the RUP Analysis and Design Discipline

The ADD method could be modeled as an activity within the Analysis and Design discipline of the RUP (see Figure 9). The participants, inputs, outputs, and function from Figure 8 correspond to the role, input artifacts, resulting artifacts, and steps in Figure 9. The software architect in the ADD method corresponds to the software architect role in the RUP.

The inputs to the ADD method, shown in Figure 8, come from other RUP artifacts. The constraints, functional requirements, and quality requirements correspond to artifacts from the requirements discipline. Constraints are also documented in the architectural proof-of-concept. Functional requirements are documented in the use case model as use cases. Also, as mentioned in the prior discussion of the QAW, quality requirements are documented in the supplementary specifications and in properties of the use cases. The ADD method mandates that these quality requirements be expressed as scenarios.

Design the Software Architecture Using the Attribute-Driven Design (ADD) Method

Purpose: The Attribute-Driven Design (ADD) Method is an approach to defining software architectures by basing the design process on the architecture's quality attribute requirements. It follows a recursive decomposition process where, at each stage in the decomposition, architectural tactics and patterns are chosen to satisfy a set of quality attribute scenarios.

Role: Software architect [Software architect]

Frequency: This activity is optional in the Inception Phase. It should occur in the first iteration of the Elaboration Phase and can recur in later iterations if substantial changes or additions to the software architecture need to be explored.

Steps:

1. Choose the module to decompose.
2. Refine the module according to these steps:
    a. Choose the architectural drivers.
    b. Choose an architectural pattern that satisfies the architectural drivers.
    c. Instantiate modules and allocate functionality from the use cases. Represent the results using multiple views.
    d. Define interfaces of the child modules.
    e. Verify and refine the use cases and quality scenarios and make them constraints for the child modules.
3. Repeat the above steps for the next module.

Input Artifacts:	**Resulting Artifacts:**
• vision [constraints] • architectural proof-of-concept [constraints] • use case model [functional requirements, quality requirements] • supplementary specifications [quality requirements]	• software architecture document [decomposition of the architecture expressed in module, concurrency, and deployment views]

Tool Mentors: None

More Information: [Bass 03]

Workflow Details:

- Analysis and Design
    - Define a Candidate Architecture
    - Perform Architectural Synthesis

Figure 9: The ADD Method as a RUP Activity[6]

[6] Correspondence to SEI terms is included in square brackets.

The outputs from the ADD method feed into other RUP activities and refine other artifacts. For example, the decomposition of the architecture corresponds to (part of) the design model artifact and the set of views captured in the software architecture document artifact in the RUP. The refined scenarios are made into or discovered as part of detailed use cases.

4.4 Reflections

The RUP fills a need in the SEI methods by placing the ADD method in a life-cycle context. The ADD method produces a course-grained architecture, and the RUP provides more guidance on how to proceed to detailed design and implementation. Incorporating the ADD Method into the RUP involves modifying the steps dealing with the high-level design of the architecture and then following the process as described by the RUP.

Similarly, ADD fills a need within the RUP: it provides a step-by-step approach for defining a candidate architecture or a more detailed architecture that can be evaluated by the ATAM or used as a blueprint for implementation. The ADD method could be modeled as an activity within the Analysis and Design discipline of the RUP (see Figure 9). Scenarios and architectural tactics are critical to architecture design. The ADD method differs from the RUP guidelines by its emphasis on addressing quality attribute requirements in an explicit way using architectural tactics. The quality attributes shape the structure of the architecture, with functionality being allocated to that structure.

The ADD method also differs from the RUP guidelines in that it provides more abstract notions of concurrency and deployment views early in the architecture design. These abstractions allow the architect greater flexibility and the opportunity to defer making more detailed decisions to a more opportune time. The candidate architecture defines, among other things, the initial layering and organization of the system, which provides a sort of module view as prescribed by the ADD method. The concurrency view prescribed by the ADD method shows conceptual threads of control and synchronization relationships among design elements that are more abstract than the notion of active classes; however, these conceptual threads are not called out in the candidate architecture. The RUP runtime architecture addresses concurrency at the more detailed level of processes and system threads. The deployment view prescribed by the ADD method shows the allocation of responsibilities to the deployment environment. Again, these responsibilities are not called out in the candidate architecture. The deployment view of the “4+1” views approach shows deployment at the more detailed level of task allocation to physical nodes.

5 Architecture Evaluation with the ATAM/CBAM

A combined Architecture Tradeoff Analysis Method (ATAM) and Cost Benefit Analysis Method (CBAM) as described by Nord [Nord 03] is best done late in the Elaboration Phase to ensure that the architecture is complete, or at the end of the Transition Phase to prepare for the next evolution cycle. This activity also highlights the risks that might be faced by further development, maintenance, and evolution.

5.1 The Integrated ATAM/CBAM

The ATAM provides software architects with a framework for understanding the technical tradeoffs and risks they face as they make design decisions. It does not provide any guidance for understanding economic tradeoffs. The CBAM helps software architects consider the ROI of any architectural decision and provides guidance on the economic tradeoffs involved. The CBAM takes the architectural decision analysis performed during an ATAM evaluation and helps make it part of a strategic roadmap for software design and evolution by associating priorities, costs, and benefits with each architectural decision.

Figure 10 provides a summary of the inputs, outputs, and participants of the combined ATAM/CBAM method. More details about the method are available in the report by Nord and colleagues [Nord 03].

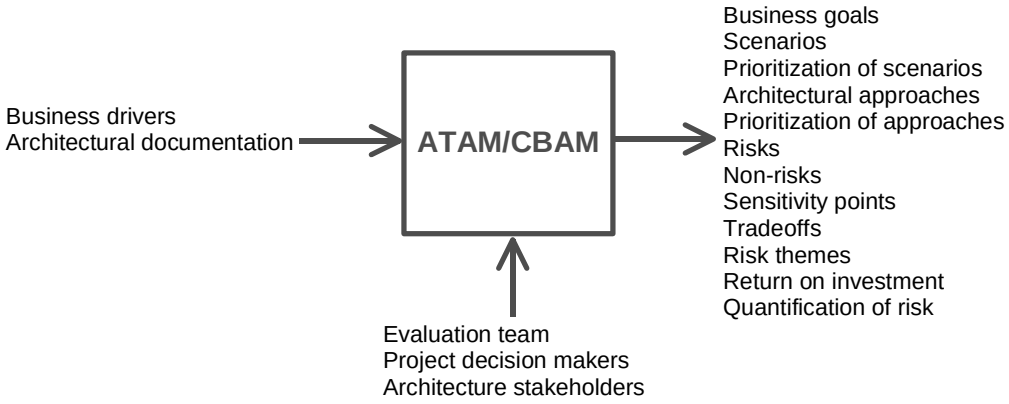

Figure 10: The Combined ATAM/CBAM Inputs, Outputs, and Participants

5.2 The ATAM/CBAM and the RUP Life Cycle

The RUP's "Review the Architecture" activity can occur at different portions of the development life cycle (as can an ATAM evaluation).

- At the end of the Inception Phase in an initial development cycle, there is usually not much of a concrete architecture in place. However, a review may uncover some unrealistic objectives, missing pieces, missed opportunities for reusing existing products, and so forth.
- The most natural place for a software architecture evaluation is at the end of the Elaboration Phase. (It is even possible to have a small evaluation at the end of each iteration in this phase.) This phase is focused primarily on exploring the requirements in detail and baselining an architecture. An architecture review is mandated by the RUP at this milestone. During this phase, a broad range of architectural qualities is examined.
- More focused evaluations may take place during the Construction Phase to examine specific quality attributes, such as performance or safety, and at the end of the Construction Phase to identify any lingering problems that may make the product unfit for delivery to its end users.
- Damage-control evaluations may take place late in the Construction or even Transition Phases, when there have been major problems (for example, construction does not complete or an unacceptable number of problems arises in the installed base during the transition).
- An evaluation may take place at the end of the Transition Phase, in particular to inventory reusable assets for an eventual new product or evolution cycle.

The ATAM is in keeping with the types of reviews categorized by the RUP: representation driven, information driven, and scenario driven. The CBAM prepares for the next maintenance iteration by considering costs and benefits, ultimately leading to a determination of the ROI. The RUP business case documents the economic value of the product, and CBAM-type reasoning can contribute to this evaluation.

5.3 The ATAM and CBAM as Activities in the RUP Analysis and Design Discipline

The RUP has a workflow defined for the Analysis and Design discipline that includes activities for reviewing the architecture. The ATAM and CBAM can be modeled as RUP activities (see Figure 11 and Figure 12).

<table>
<tr><td colspan="2">Evaluate the Software Architecture Using the Architecture Tradeoff Analysis Method (ATAM)</td></tr>
<tr><td colspan="2">Purpose: The purpose of the ATAM is to assess the consequences of architectural decisions in light of quality attribute requirements and business goals.</td></tr>
<tr><td colspan="2">Role: Technical reviewer [Evaluation team]</td></tr>
<tr><td colspan="2">Frequency: This activity occurs at least once per iteration, especially during the Elaboration Phase.</td></tr>
<tr><td colspan="2">Steps:
1. Present the ATAM.
2. Present business drivers.
3. Present architecture.
4. Identify architectural approaches.
5. Generate quality attribute utility tree.
6. Analyze architectural approaches.
7. Brainstorm and prioritize scenarios.
8. Analyze architectural approaches.
9. Present results.</td></tr>
<tr><td>Input Artifacts:
• business case, vision [business drivers]
• software architecture document [architectural documentation]
• supplementary specifications [scenarios]</td><td>Resulting Artifacts:
• review record [supplemented with risk themes and the impact they have on achieving the business goals]
• software architecture document [annotated with sensitivity points and tradeoffs]</td></tr>
<tr><td colspan="2">Tool Mentors: None</td></tr>
<tr><td colspan="2">More Information: [Bass 03]</td></tr>
<tr><td colspan="2">Workflow Details:
• Analysis and Design
  • Refine the Architecture</td></tr>
</table>

Figure 11: The ATAM as a RUP Activity[7]

[7] Correspondence to SEI terms is included in square brackets.

<table>
<tr><td colspan="2">Evaluate the ROI of Architectural Approaches Using the Cost Benefit Analysis Method (CBAM)</td></tr>
<tr><td colspan="2">Purpose: The aim of the CBAM is to explicitly associate costs, benefits, and uncertainty with architectural decisions, as a means of optimizing the choice of such decisions.</td></tr>
<tr><td colspan="2">Role: Technical reviewer [Evaluation team]</td></tr>
<tr><td colspan="2">Frequency: This activity is coupled with the ATAM.</td></tr>
<tr><td colspan="2">Steps:
1. Collate scenarios.
2. Refine scenarios.
3. Prioritize scenarios.
4. Assign intra-scenario utility.
5. Develop architectural strategies and determine quality-attribute-response levels.
6. Determine the utility of the expected quality-attribute-response levels by interpolation.
7. Calculate the total benefit obtained from an architectural strategy.
8. Choose architectural strategies based on return on investment (ROI).
9. Confirm results with intuition.</td></tr>
<tr><td>Input Artifacts:
• business case [business drivers]
• supplementary specification [scenarios]
• software architecture document [supplemented with architectural strategies]</td><td>Resulting Artifacts:
• supplementary specification [including refined scenarios]
• software architecture document [annotated with ROI determinations]</td></tr>
<tr><td colspan="2">Tool Mentors: None</td></tr>
<tr><td colspan="2">More Information: [Bass 03]</td></tr>
<tr><td colspan="2">Workflow Details:
• Analysis and Design
  • Refine the Architecture</td></tr>
</table>

Figure 12: The CBAM as a RUP Activity[8]

The participants, inputs, outputs, and function from Figure 10 correspond to the role, input artifacts, resulting artifacts, and steps in Figure 11 and Figure 12.

The ATAM/CBAM evaluation team members play the role of the RUP technical reviewer, who is responsible for contributing feedback to the review process by reviewing requirements, architecture, design, and code. The ATAM/CBAM project decision makers correspond to the RUP project manager and software architect. The ATAM/CBAM architecture stake-

[8] Correspondence to SEI terms is included in square brackets.

holders correspond to workers in the various disciplines and a business team. These stakeholders include the systems analyst and the requirements specifier for the requirements discipline, and the software architect, designer, user-interface designer, database designer, capsule designer, and test designer in the Inception Phase. The systems analyst produces the requirements management plan, the glossary, quality attributes, use case models, supplementary specifications, stakeholder requests, the vision document, and a user-interface prototype, either by storyboard or code. The requirements specifier is in charge of use cases for the software requirements specification, a use case package, and the software requirements document. In the Elaboration Phase, the architect produces the deployment model, software architecture document, analysis model, design model, architectural proof-of-concept, and reference architecture, and the interface, signals, events, and protocols for real-time systems. The capsule designer produces the timing capsule, if needed; the designer produces the detailed design of the architectural components.

The inputs to the combined ATAM/CBAM come from other RUP artifacts. The ATAM/CBAM business drivers describe the system's most important functions (any relevant technical, managerial, economic, or political constraints; the business goals and context as they relate to the development project; and the major stakeholders) and the architectural drivers (that is, the major quality attribute goals that shape the architecture). This information can come from the RUP business vision artifact (from the business-modeling discipline) and the software requirements specification (consisting of use cases and the supplementary specification).

The ATAM/CBAM architectural documentation describes the driving architecture requirements and important architectural information: a context diagram; module or layer view; component-and-connector view; deployment view; and the architectural approaches, patterns, or tactics employed, including which quality attributes are addressed and how. These requirements come from the RUP's software architecture document and are supported by the guidelines for using architectural views.

The outputs from the combined ATAM/CBAM may feed into other RUP activities and/or refine other artifacts. For example, business goals are elicited or reviewed during the ATAM and could be fed back to the RUP business vision. The scenarios can help refine the supplementary specifications. Sensitivity points, tradeoffs, and the ROI provide enhanced documentation for the architecture. Risks and risk themes can provide feedback to the business goals and the project management activities. All of these outputs from the combined ATAM/CBAM provide feedback to the architect to make educated design decisions.

5.4 Reflections

The RUP fills a need in the architecture-centric methods by placing the ATAM/CBAM in a life-cycle context. It places the ROI computed by the CBAM in the larger context of project and product costs and benefits as documented in the business case—a project management artifact used to make go/no-go decisions at key milestones, such as the Life-Cycle Objective and Life-Cycle Architecture Milestones.

The ATAM is an architecture evaluation method that fits in with the RUP by defining a step-by-step approach to evaluating a software architecture. The RUP has a review record artifact that records problems and acts as a placeholder for the risks uncovered by the ATAM. But the ATAM adds additional value in producing risk themes and showing the impact they have on achieving the business goals. The ATAM makes the evaluation of decisions to accommodate quality attribute requirements explicit. The ATAM also contributes artifacts not necessarily found in the review record. Sensitivity points and tradeoffs provide enhanced documentation for the architecture, concentrating on areas where risk is potentially highest. Scenarios provide feedback for existing and future requirements. These ATAM artifacts all contribute toward improving the architecture.

The CBAM provides more details on the business consequences of architecture decisions implied by the architecture, allowing the architect to make informed choices among architectural options.

6 Architecture Evaluation with ARID

Software architectures often consist of complicated subdesign problems. If these intermediate designs are inappropriate, the architecture can be undermined. Active Reviews for Intermediate Designs (ARID) is a lightweight evaluation approach that can be used to examine a design as it is developed and before it is released. For example, it may be used in the Elaboration Phase as the overall architecture is being designed to determine the design's viability.

6.1 ARID

ARID is a scenario-based, stakeholder-centric review of a portion of a software architecture, typically, a coherent software-invocable service. ARID blends Active Design Reviews [Parnas 01] with the ATAM, creating a technique for investigating partially completed designs. The review is focused on whether the design is sufficient to support the software developers who will use it. ARID helps to find issues and problems that hinder the successful use of the design as it has been conceived.

Figure 13 provides a summary of ARID's inputs, outputs, and participants. Clements describes ARID in more detail [Clements 00].

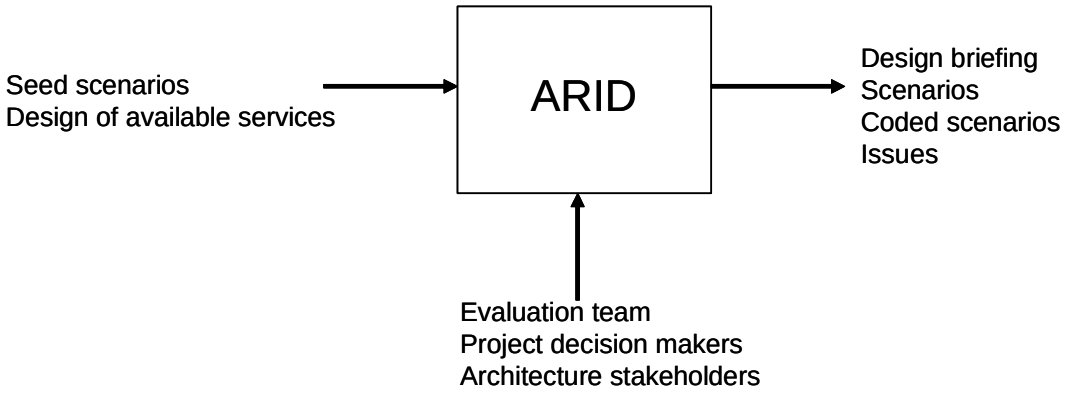

Figure 13: ARID's Inputs, Outputs, and Participants

6.2 ARID and the RUP Life Cycle

The RUP has a workflow defined for analysis and design that includes activities for reviewing the architecture. ARID is used for reviewing components of that architecture; it represents a combination of analysis approaches: representation driven, information driven, and scenario driven. It could occur as a form of architecture review during an iteration within the Elaboration Phase. An ARID could also occur at the end of the Elaboration Phase following an ATAM evaluation for more detailed analysis. Finally, ARID could occur as part of a design review in the Construction Phase.

6.3 ARID as an Activity in the RUP Analysis and Design Discipline

ARID is modeled as an activity within the analysis and design discipline of the RUP (see Figure 14). The participants, inputs, outputs, and function from Figure 13 correspond to the role, input artifacts, resulting artifacts, and steps in Figure 14.

Workers such as the designers and system architects are the primary stakeholders for the review. The inputs to ARID come from other RUP artifacts, such as parts of the candidate or executable architecture. The outputs from ARID feed into other RUP activities, such as defining or refining the executable architecture and/or refining other artifacts.

<table>
<tr><td colspan="2">Evaluate Architectural Service Using Active Reviews for Intermediate Designs (ARID)</td></tr>
<tr><td colspan="2">Purpose: ARID is a scenario-based, stakeholder-centric review of a portion of an architecture, typically, a coherent software-invocable service. The review is focused on whether the design is sufficient for the software developers who will use it.</td></tr>
<tr><td colspan="2">Role: Technical reviewer [Evaluation team, Architecture stakeholder]</td></tr>
<tr><td colspan="2">Frequency: This activity occurs at least once per iteration, especially during the Elaboration Phase. It is optional during the Construction Phase for detailed analysis of the interface specifications.</td></tr>
<tr><td colspan="2">Steps:
1. Identify the reviewers.
2. Prepare the design briefing.
3. Prepare the seed scenarios.
4. Prepare the materials.
5. Present ARID.
6. Present the design.
7. Brainstorm and prioritize scenarios.
8. Apply the scenarios.
9. Summarize.</td></tr>
<tr><td>Input Artifacts:
• software architecture document [design of available services]
• supplementary specifications [seed scenarios]</td><td>Resulting Artifacts:
• review record [annotated with issues]</td></tr>
<tr><td colspan="2">Tool Mentors: None</td></tr>
<tr><td colspan="2">More Information: [Clements 02a]</td></tr>
<tr><td colspan="2">Workflow Details:
• Analysis and Design
  • Refine the Architecture
  • Analyze Behavior</td></tr>
</table>

Figure 14: The ARID Method as a RUP Activity[9]

[9] Correspondence to SEI terms is included in square brackets.

6.4 Reflections

The RUP fills a need in the architecture-centric methods by placing ARID in a life-cycle context. ARID provides an architectural evaluation method, but only for the specific components of the architecture that are investigated in greater detail.

7 Summary

In this report, we have summarized the RUP and shown where specific SEI architecture-centric methods can help to produce artifacts required in different RUP phases. Table 1 shows the highlights.

Method	Role	Discipline	Workflow Detail	Artifacts Affected
QAW	Systems analyst	Requirements	Understand Stakeholder Needs	• Business case • Supplementary specifications
ADD	Software architect	Analysis & Design	Define a Candidate Architecture Perform Architectural Synthesis	Software architecture document
ATAM/ CBAM	Technical reviewer	Analysis & Design	Refine the Architecture	• Review record • Software architecture document
ARID	Technical reviewer	Analysis & Design	Refine the Architecture Analyze Behavior	Review record

Table 1: The Architecture-Centric Methods as RUP Activities

We have also shown how specific SEI methods can enhance the activities and artifacts of the RUP, thus enhancing the value of the RUP as a design process. The benefit of including the SEI methods is to address quality attributes in an explicit, methodical, engineering-principled way. We believe that quality attribute requirements drive the software architecture and that architecture-centric activities (with an explicit focus on quality attributes) drive the software system life cycle.

The RUP artifacts from the Initiation and Elaboration Phases can benefit the most from the application of SEI methods. This fact implies that the architecture-centric methods can be done early in the RUP life cycle (which we have always claimed). However, as with all such methods, the RUP, augmented with architecture-centric methods, is a “garbage-in, garbage-out” process. The willing participation of the appropriate stakeholders is crucial to the success of such methods. Properly managed, the architecture-centric methods can be a low-cost addition to the RUP that will dramatically increase the quality of the systems and products developed.

References

URLs are valid as of the publication date of this document.

[Bachmann 02] Bachmann, F.; Bass, L.; & Klein, M. *Illuminating the Fundamental Contributors to Software Architecture Quality* (CMU/SEI-2002-TR-025, ADA407778). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tr025.html

[Barbacci 03] Barbacci, M. R.; Ellison, R.; Lattanze, A. J.; Stafford, J. A.; Weinstock, C. B.; & Wood, W. G. *Quality Attribute Workshops (QAWs), Third Edition* (CMU/SEI-2003-TR-016, ADA418428). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html

[Bass 03] Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice, Second Edition.* Boston, MA: Addison-Wesley, 2003.

[Boehm 96] Boehm, B. "Anchoring the Software Process." *IEEE Software 13*, 4 (July 1996): 73-82.

[Cantor 03] Cantor, M. "Rational Unified Process for Systems Engineering." *Rational edge: the e-zine for the rational community* (September 2003). http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/sep03/m_systemarch_mc.pdf

[Clements 00] Clements, P. *Active Reviews for Intermediate Designs* (CMU/SEI-2000-TN-009, ADA383775). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications/documents/00.reports/00tn009.html

[Clements 02a] Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies*. Boston, MA: Addison-Wesley, 2002.

[Clements 02b] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond*. Boston, MA: Addison-Wesley, 2002.

[Grady 92] Grady, R. *Practical Software Metrics for Project Management and Process Improvement.* Englewood Cliffs, NJ: Prentice-Hall, 1992.

[IBM 04] IBM. *Rational Unified Process*. http://www-306.ibm.com /software/awdtools/rup/ (July 2004).

[IEEE 98] Institute of Electrical and Electronics Engineers. *IEEE Standard for Functional Modeling Language* (IEEE Std 1320.1-1998). New York, NY: IEEE Computer Society, 1998 (ISBN 0-738-10340-3).

[Jacobson 99] Jacobson, Ivar; Booch, Grady; & Rumbaugh, James. *The Unified Software Development Process*. Boston, MA: Addison-Wesley, 1999.

[Kazman 96] Kazman, R.; Abowd, G.; Bass, L.; & Clements, P. "Scenario-Based Analysis of Software Architecture." *IEEE Software 13*, 6 (Nov. 1996): 47-55.

[Kazman 00] Kazman, R.; Klein, M.; & Clements, P. *ATAM: Method for Architecture Evaluation* (CMU/SEI-2000-TR-004, ADA382629). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2000. http://www.sei.cmu.edu/publications /documents/00.reports/00tr004.html

[Kazman 02] Kazman, R.; Asundi, J.; & Klein, M. *Making Architecture Design Decisions: An Economic Approach* (CMU/SEI-2002-TR-035, ADA408740). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu /publications/documents/02.reports/02tr035.html

[Kazman 03] Kazman, R.; Nord, R. L.; & Klein, M. *A Life-Cycle View of Architecture Analysis and Design Methods* (CMU/SEI-2003-TN-026, ADA421679). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tn026.html

[Kroll 03] Kroll, Per & Kruchten, Philippe. *The Rational Process Made Easy: A Practitioner's Guide to the RUP*. Boston, MA: Addison-Wesley, 2003.

[Kruchten 95] Kruchten, P. "The 4+1 View Model of Architecture." *IEEE Software 12*, 6 (November 1995): 42-50.

[Kruchten 04] Kruchten, P. *The Rational Unified Process: An Introduction, Third Edition*. Boston, MA: Addison-Wesley, 2004.

[Leffingwell 00] Leffingwell, Dean & Widrig, Don. *Managing Software Requirements.* Boston, MA: Addison-Wesley, 2000.

[Nord 03] Nord, R.; Barbacci, M.; Clements, P.; Kazman, R.; O'Brien, L.; & Tomayko, J. *Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method* (CBAM) (CMU/SEI-2003-TN-038, ADA421615). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. http://www.sei.cmu.edu/publications/documents/03.reports/03tn038.html

[Parnas 01] Parnas, D. & Weiss, D. Ch. 17, "Active Design Reviews," 337-351. *Software Fundamentals: Collected Papers by David L. Parnas*. Boston, MA: Addison-Wesley, 2001.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	July 2004	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Integrating Software-Architecture-Centric Methods into the Rational Unified Process	F19628-00-C-0003

6. AUTHOR(S)

Rick Kazman, Philippe Kruchten, Robert L. Nord, James E. Tomayko

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2004-TR-011

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2004-011

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The Rational Unified Process (RUP) is used broadly by software developers. This technical report fits the Carnegie Mellon® Software Engineering Institute's (SEI's) architecture-centric methods into the framework of the RUP. These methods include the Architecture Tradeoff Analysis Method®, the SEI Quality Attribute Workshop, the SEI Attribute-Driven Design method, the SEI Cost Benefit Analysis Method, and SEI Active Reviews for Intermediate Design. Since the key process milestone of the Elaboration Phase of the RUP is a completed architecture, the architecture-centric methods appear early in the process during the first two phases (i.e., Inception and Elaboration). This report presents a summary of the RUP and then examines the potential uses of the SEI's architecture-centric methods.

14. SUBJECT TERMS	15. NUMBER OF PAGES
architecture-centric methods, Architecture Tradeoff Analysis Method, ATAM, Active Reviews for Intermediate Design, ARID, Attribute-Driven Design method, ADD method, Cost Benefit Analysis Method, CBAM, Rational Unified Process, RUP	50

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL